

Devang Patel Institute of Advance Technology and Research

Department of Computer Engineering





14

Practical - 4

Aim: Greedy Approach

4.1) A Burglar has just broken into the Fort! He sees himself in a room with n piles of gold dust. Because each pile has a different purity, each pile also has a different value (v[i]) and a different weight (w[i]). A Burglar has a bag that can only hold W kilograms. Calculate which piles Burglar should completely put into his bag and which he should put only fraction into his bag. Design and implement an algorithm to get maximum piles of gold using given bag with W capacity, Burglar is also allowed to take fractional of pile.

Program Code:

```
import java.util.*;
public class prac {

private static class Item {
    int value, weight;
    double ratio;

public Item(int value, int weight) {
        this.value = value;
        this.weight = weight;
        this.ratio = (double) value / weight;
    }

private static void sortItems(Item[] items) {
    for (int i = 0; i < items.length - 1; i++) {
        for (int j = i + 1; j < items.length; j++) {
        if (items[i].ratio < items[j].ratio) {
}</pre>
```



Devang Patel Institute of Advance Technology and Research



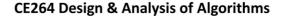


```
Item temp = items[i];
          items[i] = items[j];
          items[j] = temp;
       }
     }
  }
}
private static double getMaxValue(int[] values, int[] weights, int capacity) {
  int n = values.length;
  Item[] items = new Item[n];
  for (int i = 0; i < n; i++) {
    items[i] = new Item(values[i], weights[i]);
  sortItems(items);
  double total Value = 0;
  int currentWeight = 0;
  for (Item item: items) {
    if (currentWeight + item.weight <= capacity) {</pre>
       totalValue += item.value;
       currentWeight += item.weight;
     } else {
       int remainingCapacity = capacity - currentWeight;
       totalValue += (double) item.value * remainingCapacity / item.weight;
       break;
     }
  return totalValue;
```



Devang Patel Institute of Advance Technology and Research

Department of Computer Engineering





```
public static void main(String[] args) {
    int[] values = {60, 100, 120};
    int[] weights = {10, 20, 30};
    int capacity = 50;
    double maxValue = getMaxValue(values, weights, capacity);
    System.out.println("Maximum value of gold dust that the burglar can steal: " + maxValue);
}
```

Output:

```
PS D:\Probin's Work\Extra> javac prac.java
PS D:\Probin's Work\Extra> java prac
Maximum value of gold dust that the burglar can steal: 240.0
PS D:\Probin's Work\Extra>
```

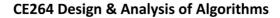
4.2) Implement the program to find the shortest path from one source to all other destinations in any city graph.

Program Code:

```
import java.util.*;
public class prac {
  private int V;
  private List<List<Node>> adj;
  public prac(int V) {
    this.V = V;
    adj = new ArrayList<>(V);
    for (int i = 0; i < V; i++) {
        adj.add(new ArrayList<>());
    }
}
```



Devang Patel Institute of Advance Technology and Research

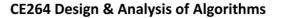




```
}
}
class Node implements Comparable<Node> {
  int dest;
  int weight;
  Node(int dest, int weight) {
    this.dest = dest;
    this.weight = weight;
  }
  public int compareTo(Node other) {
    return Integer.compare(this.weight, other.weight);
  }
}
public void addEdge(int source, int dest, int weight) {
  adj.get(source).add(new Node(dest, weight));
  adj.get(dest).add(new Node(source, weight));
}
public void shortestPath(int source) {
  PriorityQueue<Node> pq = new PriorityQueue<>();
  int[] dist = new int[V];
  Arrays.fill(dist, Integer.MAX_VALUE);
  pq.add(new Node(source, 0));
  dist[source] = 0;
```



Devang Patel Institute of Advance Technology and Research



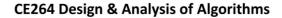


```
while (!pq.isEmpty()) {
     int u = pq.poll().dest;
     for (Node neighbor : adj.get(u)) {
       int v = neighbor.dest;
       int weight = neighbor.weight;
       if (dist[u] + weight < dist[v]) {</pre>
          dist[v] = dist[u] + weight;
          pq.add(new Node(v, dist[v]));
       }
     }
  System.out.println("Shortest paths from source " + source + " to all destinations:");
  for (int i = 0; i < V; i++) {
     System.out.println("Destination " + i + ": " + dist[i]);
  }
}
public static void main(String[] args) {
  int V = 5;
  prac graph = new prac(V);
  graph.addEdge(0, 1, 2);
  graph.addEdge(0, 3, 6);
  graph.addEdge(1, 2, 3);
  graph.addEdge(1, 3, 8);
  graph.addEdge(1, 4, 5);
  graph.addEdge(2, 4, 7);
  graph.addEdge(3, 4, 9);
```



Devang Patel Institute of Advance Technology and Research

Department of Computer Engineering





```
graph.shortestPath(0);
}
```

Output:

}

```
PS D:\Probin's Work\Extra> javac prac.java
PS D:\Probin's Work\Extra> java prac
Shortest paths from source 0 to all destinations:
Destination 0: 0
Destination 1: 2
Destination 2: 5
Destination 3: 6
Destination 4: 7
PS D:\Probin's Work\Extra>
```

4.3) Find Minimum Cost spanning tree of an undirected graph using Kruskal's algorithm.

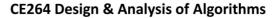
Program Code:

```
import java.util.*;
class Edge {
  int source, dest, weight;

Edge(int s, int d, int w) {
    source = s;
    dest = d;
    weight = w;
  }
} class Graph {
```



Devang Patel Institute of Advance Technology and Research

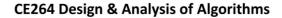




```
int V, E;
Edge[] edges;
Graph(int v, int e) {
  V = v;
  E = e;
  edges = new Edge[E];
  for (int i = 0; i < e; ++i) {
     edges[i] = new Edge(0, 0, 0);
  }
}
void sortEdges() {
  for (int i = 0; i < E - 1; i++) {
     for (int j = 0; j < E - i - 1; j++) {
        if (edges[j].weight > edges[j + 1].weight) {
          Edge temp = edges[j];
          edges[j] = edges[j + 1];
          edges[j + 1] = temp;
     }
   }
int find(int[] parent, int i) {
  if (parent[i] == i) return i;
  return find(parent, parent[i]);
}
void union(int[] parent, int[] rank, int x, int y) {
```



Devang Patel Institute of Advance Technology and Research





```
int xRoot = find(parent, x);
  int yRoot = find(parent, y);
  if (rank[xRoot] < rank[yRoot]) {</pre>
     parent[xRoot] = yRoot;
  } else if (rank[xRoot] > rank[yRoot]) {
     parent[yRoot] = xRoot;
  } else {
     parent[yRoot] = xRoot;
     rank[xRoot]++;
  }
}
void kruskalMST() {
  Edge[] result = new Edge[V];
  int e = 0;
  int i = 0;
  for (i = 0; i < V; ++i) {
     result[i] = new Edge(0, 0, 0);
  }
  sortEdges();
  int[] parent = new int[V];
  int[] rank = new int[V];
  for (i = 0; i < V; ++i) {
     parent[i] = i;
     rank[i] = 0;
```

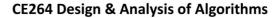


i = 0;

Charotar University of Science and Technology

Devang Patel Institute of Advance Technology and Research

Department of Computer Engineering





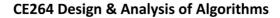
```
while (e < V - 1) {
       Edge nextEdge = edges[i++];
       int x = find(parent, nextEdge.source);
       int y = find(parent, nextEdge.dest);
       if (x != y) {
          result[e++] = nextEdge;
          union(parent, rank, x, y);
       }
     }
     System.out.println("Minimum Cost Spanning Tree: ");
     int minimumCost = 0;
     for (i = 0; i < e; ++i) {
       System.out.println(result[i].source + " - " + result[i].dest + ": " + result[i].weight);
       minimumCost += result[i].weight;
     System.out.println("Minimum Cost: " + minimumCost);
  }
public class prac {
  public static void main(String[] args) {
    int V = 4;
    int E = 5;
     Graph graph = new Graph(V, E);
     graph.edges[0] = new Edge(0, 1, 10);
     graph.edges[1] = new Edge(0, 2, 6);
     graph.edges[2] = new Edge(0, 3, 5);
```

}



Devang Patel Institute of Advance Technology and Research

Department of Computer Engineering





```
graph.edges[3] = new Edge(1, 3, 15);
graph.edges[4] = new Edge(2, 3, 4);
graph.kruskalMST();
}
```

Output:

}

```
PS D:\Probin's Work\Extra> javac prac.java
PS D:\Probin's Work\Extra> java prac
Minimum Cost Spanning Tree:
2 - 3: 4
0 - 3: 5
0 - 1: 10
Minimum Cost: 19
PS D:\Probin's Work\Extra>
```

Conclusion: From this practical I learned how to use greedy approach for getting the best optimal solution.

Staff Signature:

Grade:

Remarks by the Staff: